

Extending Your Apps

BY NEIL J. RUBENKING

The Structure of a Help System

It's time to update the old saying "The job isn't over till the paperwork is done." When it comes to writing Windows applications, the job isn't truly over till the help system is done. A printed manual is no longer sufficient or even necessary, but on-line help is a must—and context-sensitive help is even better. Fortunately, modern tools make help-system creation simple, so there's just no excuse for skipping this step.

Even the best tools won't do the work for you, though. To build a decent help system, you need to write clear and simple text that explains how your program works. You also need to program the connections between the topics of your help system. Most of all, though, you need to understand all the elements of a good help system. You can be the world's best programmer and still produce an inadequate help system if you aren't familiar with all the features WINHELP.EXE (the built-in Windows help engine) offers. This article will explain the elements of a help system, both simple and advanced. It won't discuss the nitty-gritty details of help-system construction, since those will vary depending on the design tool that you use.

Figure 1 is a screenshot of a sample help system you can refer to as you read this article. The sample help system was designed strictly to show as many help features as possible—it's not meant as an example of good layout and design!

WRITING HELP TOPICS Just as a book is divided into pages, a help system is divided into topics. A topic should cover a single concept, and most topics should contain no more than one screenful of text. Every topic has a title and a context

string. From the user's point of view, the title is the name of the topic. For example, when WinHelp's Search dialog displays a list of topics, what it shows is the titles of the topics. Each topic's unique context string identifies the topic to WinHelp internally. Common practice is to make the context string the same as the title, replacing any spaces with underscores—many help-creation tools do this automatically. You can optionally assign a unique context number to each topic. Context numbers are useful when working with programming environments like Microsoft Visual Basic, in which menu choices and on-screen controls can be linked directly to help-context numbers.

Frequently one topic will refer to information that's covered in more detail in another topic. A printed manual would use a reference like "(see page 27)"; a help system can incorporate a jump directly to the other topic. A word or phrase is designated as the hot spot for the jump—the help system underlines the hot-spot text and, by default, displays it in green. Jumps are sometimes called links or hypertext links. When the user clicks on a jump hot spot, the help system switches to the specified target page. To return from one or several jumps, the user simply clicks WinHelp's Back button.

A printed manual might insert the definition of an unfamiliar word or phrase as a footnote. Help systems use a pop-up topic instead. The word or phrase designated as the pop-up hot spot is displayed in green by default, with a dotted underline.

Clicking on the pop-up hot spot doesn't switch to a new topic. Rather, it causes the text of the pop-up topic to appear in a window that seems to float above the main WinHelp window. Another mouse click disposes of the pop-up window. As discussed below, bitmap images in the help system can also be hot spots for jumps or pop-ups.

Note that although the text of jump and pop-up hot spots is green by default, the user can choose different colors. Adding a JumpColor key to the [Windows Help] section of WIN.INI changes the color of both jumps and pop-ups; adding a PopupColor key sets the color of pop-ups alone. The value for each of these keys consists of three numbers from 0 to 255 indicating the amount of red, green, and blue in the color. For example:

```
JumpColor=192 0 0
PopupColor=0 0 192
```

would cause WinHelp to display jump hot spots in a medium red and pop-ups in a medium blue.

NAVIGATING THE HELP SYSTEM Every help system must have a Contents topic; it's a kind of "home base" for the help system, and its title is usually just "Contents" or "<program name> Help Contents." This is the topic that WinHelp displays when no specific topic is requested, and WinHelp's Contents button jumps straight to this topic. Usually the Contents topic contains almost nothing but jumps

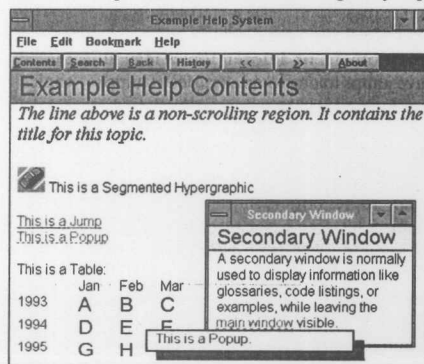


Figure 1: Most of the elements of this sample help system explain themselves. The About button and the browse buttons were added using help macros.

■ Snapping Your Screens

You don't need a fancy screen-capture utility to take a snapshot of your program's main window. Run your program and press Alt-PrtSc to copy the active window to the clipboard. Launch Paintbrush and maximize it, then paste in the contents of the clipboard. (If the image is too big, try removing the check marks next to Tools and Linesize and Palette under the View menu and paste again.) Now select Cursor Position from the View menu, and place the mouse cursor precisely at the bottom right corner of the image. Note the x,y coordinates in the cursor-position window.

From the Options menu, select

Image Attributes. In the resulting dialog box, click on the "pels" radio button (*pels* means pixels), and fill in the Width and Height boxes with the noted coordinates of the image's bottom right corner. Click OK, answer Yes to the warning that the current image will be lost, and answer No to the query about saving current changes. Now paste the clipboard into Paintbrush again. If you've done everything right, the image will precisely fit the working area of Paintbrush. Select Save As from the File menu, and set the file type to "16 color bitmap." Choose a name and save the file. You've snapped the screen!

to other high-level topics. You may want to think of the Contents topic as being equivalent to the table of contents in a book, and the jumps the Contents topic contains as equivalent to the beginnings of chapters. A simple Contents topic might contain jumps to topics explaining each of the program's windows and each of the top-level menu items. The user will see the Contents topic a lot, so it should be orderly and attractive. Every single additional topic should be accessible, via one route or another, from the Contents topic.

The jumps in a well-written help system form a complicated web of connections. One topic may include jumps to a dozen others, or a dozen topics may all have jumps to one topic. A series of jumps may form a loop, bringing you back to the starting topic. For example, in WinHelp's own help system the "Searching for a Help Topic" and "Moving Around in Help" topics both include jumps to each other. This free-association network is convenient for the user who's pursuing a particular train of thought, but it's not much use when you want to "skim" the help system. A good help system will also include one or more browse sequences (see Figure 2). A *browse sequence* is a linear series of related help topics intended to be read in order. Each topic can belong to one and only one browse sequence.

When the current topic belongs to a browse sequence, WinHelp's browse buttons (marked with << and >>) are enabled. Pressing a browse button flips to the previous or next topic in the current browse sequence.

How do you decide which topics to link in a browse sequence? It's up to you, of course, but here are some guidelines. In a very small help system, you'll create a single browse sequence that includes every topic. Start with the Contents topic and add the remaining topics in any order that makes sense to you. In a larger help system, start a browse sequence with each of the "chapter heading" topics reached by a single jump from the Contents topic. Again, arrange the topics in an order that makes sense to you, and make sure that every topic is included in one browse sequence.

Keywords are words and phrases that serve as the help system's index. Each topic can have any number of associated keywords, though more than three or four is unusual. WinHelp's Search dialog initially displays a list of all keywords in the

lect one, the lower list box fills with a list of all topics associated with that keyword (see Figure 3). Keywords give the user a quick way to dive into a particular topic. Note that a keyword need not appear literally in the topic's text.

Assigning keywords to topics can be a tedious task, but it's extremely important. Pick descriptive words and phrases for keywords, and assign at least one keyword to every topic. If nothing else, use the topic's title as a keyword. It's very important to be consistent in your keywords. Don't use "open files" for one topic and "open file" for another, for example. Some help-design tools make it easy to reuse existing keywords. If yours doesn't, it's a good idea to keep a list of the keywords you've already added to the system, and reuse them when possible.

WORKING WITH GRAPHICS Windows is a graphical user interface, and Windows help systems are definitely improved by the judicious use of bitmapped graphics. Graphics may be purely decorative, like a company logo on the Contents topic, or they may be an active part of the help system with embedded hot spots. The technical term for bitmaps that include hot spots is *segmented hypergraphics*, and Microsoft's freely available SHED editor can turn an ordinary bitmap into a segmented hypergraphic (.SHG) file. A graphical hot spot is linked to its topic via the topic's context string. Modern help

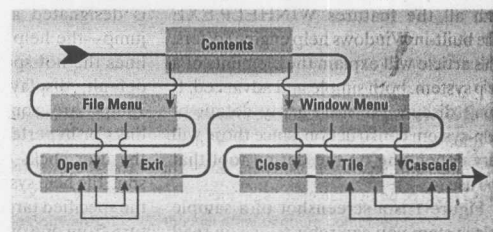


Figure 2: In this simple help system, jumps (blue arrows) make a web of connections between the topics. The browse sequence (red arrow) is a linear path through the help system that visits every topic exactly once.

systems often incorporate a bitmap of the program's main screen, in which a graphic hot spot for each on-screen element of the main window jumps to a help topic explaining that element.

Choosing Your Tools

It's possible to build a help system using just three tools: a Windows word processor that can generate .RTF (Rich Text Format) files, the Windows Help Compiler, and the SHED graphics editor. You can download the latter two from CompuServe's WINSKD forum, Library 16, as HCP.ZIP and SHED.ZIP. But writing a help system this way is as tedious and counterproductive as assembling the bytes of your Windows application using DEBUG. Modern tools such as RoboHelp, from Blue Sky Software Corp. (reviewed in "Word Processing Utilities: RoboHelp Uses Word to Build Win-

dows Help Files," *PC Magazine*, August 1994), and Doc-to-Help, from WexTech Systems ("Word Processor Add-Ins," February 9, 1993), work with your word processor to generate the necessary RTF file. Other tools such as The Windows Help Magician, from Software Interphase ("Create Help Files Quickly With The Help Magician," July 1993), and ForeHelp, from ForeFront ("ForeHelp: Help-File Heaven," October 25, 1994), provide their own WYSIWYG help editors. This is not an exhaustive list; there are help-design tools to suit every need and budget.

In many cases, the tools Windows itself provides are sufficient to let you create a bitmap of your program's main window, as the sidebar "Snapping Your Screens" explains. Regardless of whether or not you intend to create graphical hot spots in the bitmap, you should load it into the SHED editor and save it as an .SHG file. Why? Because the .SHG file format uses compression, and the .BMP format does not. An .SHG file can be substantially smaller than a corresponding .BMP file. To create a graphical hot spot in the SHED editor, you use the mouse to drag a rectangle that defines the hot spot area, and you associate the hot spot area with the context string of the corresponding help-system topic as shown in Figure 4.

Bitmaps make your help file bigger, even when stored as compressed .SHG files. Don't overuse them! For a simple program, one picture of the main window may be enough. And remember that although help text in topics wraps when the help window is resized, bitmaps can't. Always use the smallest bitmap that will do.

USING HELP MACROS A basic help system consists of help topics linked by jumps and browse sequences, with a set of keywords to serve as an index. It may include bitmapped graphics as well. However, to go beyond the basics you'll need to learn about help macros. The jump and pop-up hot spots that you embed in help

topics are effectively commands to WinHelp, telling it to jump to a particular topic or display a pop-up window. Help macros are another kind of command for WinHelp. There are over 50 macros that can be programmed to activate when the help system loads, when a particular topic is activated, or when the user clicks on a hot spot. This article won't attempt an ex-

haustive discussion of each help macro; what's important is that you understand the kinds of commands that are available. A dozen or so of the macro commands let you create, delete, enable, disable, and change the effect of a WinHelp menu item or button. The result (called the

```
CreateButton ('About', '&About',  
'About()')
```

binding) of activating a menu item or button is always a macro. Another 20-odd macro commands duplicate the effect of WinHelp's built-in menu and button commands. For example, the Next macro switches to the next topic in the current browse sequence, and the About macro displays WinHelp's About box. In the sample help system shown in Figure 1, this macro creates the About button and binds it to the About macro:

Yet another group of macros gives complete control over jumps and pop-ups within this file or into any other help file. Possibly the most common macro command is BrowseButtons, which adds the standard browse buttons to WinHelp's button bar and is usually activated at the time the help system loads. Your help system should include browse sequences, so you'll definitely need to use BrowseButtons. Some help-design tools automatically include the BrowseButtons macro when any browse sequences are present.

Certainly the flashiest WinHelp macro commands are ExecProgram and RegisterRoutine. ExecProgram launches the Windows application of your choice. You might, for example, create a button that activates a computer-based training program using ExecProgram. RegisterRoutine gives your help system access to any Windows API function or any function defined in a DLL (Dynamic Link Library). Once a function is registered, you call it just as if it were another macro. RegisterRoutine is often used to add multimedia support to help files: for example, to play a .WAV file for pronunciation when the user clicks on an unusual word.

DETAILING THE HELP SYSTEM Expert help-system designers can add a number of refinements. Give a finishing touch to the text in your help topics by using multiple fonts and font effects such as bold, italic, and underline. Since you don't know what fonts your users will have installed,

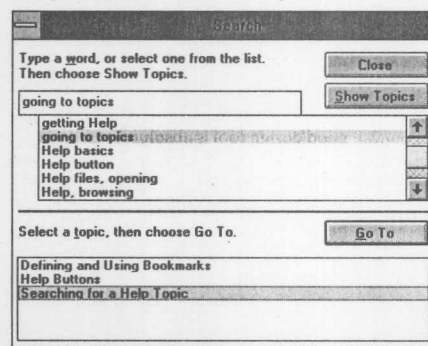


Figure 3: The top list box of WinHelp's Search dialog lists all keywords in the help system; the bottom list box shows all topics related to the selected keyword.

haustive discussion of each help macro; what's important is that you understand the kinds of commands that are available.

A dozen or so of the macro commands let you create, delete, enable, disable, and change the effect of a WinHelp menu item or button. The result (called the

VB Power



ISBN: 1-56276-073-4
Price: \$34.95

This innovative guide by Dan Appleman turns Visual Basic programmers into power users by illuminating the tools of the Windows Application Program Interface (API). Includes a time-saving disk that contains source code from the examples in the text, plus a specially designed dynamic link library (DLL).

Available at fine bookstores, or call
1-800-688-0448
ext. 8448



© 1993 Ziff-Davis Press

PC TECH
Extending Your Apps

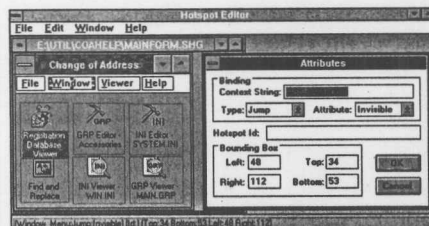


Figure 4: In this sample bitmap of a program's main window, each menu item and icon is designated as a graphical hot spot. The Attributes dialog box is displaying details about the window-menu hot spot.

you should only use the fonts Microsoft Windows 3.1 loads by default: Arial, Courier New, MS Serif, MS Sans Serif, Symbol, Times New Roman, and Wingdings.

Topic text wraps to fit the width of the WinHelp window. If it won't all fit, a vertical scroll bar appears. Once the user scrolls into the topic, the topic heading isn't visible! If you put the topic heading in a non-scrolling region, it won't disappear when the text scrolls. You can give the non-scrolling region its own background color too, as Figure 1 shows.

If your help system needs to present tabular data, don't torture yourself trying to line up items using Tab characters. WinHelp includes direct support for tables. But creating a table "by hand" using WinHelp codes is incredibly tedious and difficult. This is one case where a good design tool is absolutely essential.

In some instances it makes sense to enhance your help system with one or more secondary windows. For example, the help system in Microsoft Word for Windows, Version 6.0 uses a secondary window for "How To" topics, and the help system for many programming environments uses a secondary window to show examples of the use of various program elements. Figure 1 includes a secondary window; note that the secondary window has no menu or button bar.

HOW SENSITIVE SHOULD YOU BE? Your program's help system need not be context-sensitive. As long as it includes the standard help-menu items, the user can instantly view the Contents topic or search for help on a given keyword. If the help system is well organized, context-sensitive help may not be necessary. On

the other hand, if the development environment you're using has support for context-sensitive help built in, you can give every window, every menu item, and every on-screen control its own personal help topic.

Precisely how context-sensitive you make your help system is a case-by-case matter and can't be defined by rules. My own suggestion, however, is a compromise between no context-sensitive help and a topic for every push button. If your

program has multiple windows and dialog boxes, devote a help topic to each of these, and write your code so that pressing F1 (the standard help key) brings up the help topic for the current window. For menu items and controls, display a single-line "hint" on the program's status bar for the menu item or control that currently has the focus. If your development environment supports it, create "flyover help" or "tooltips" for toolbar items. *Flyover help*, as shown in Figure 5, is a tiny box

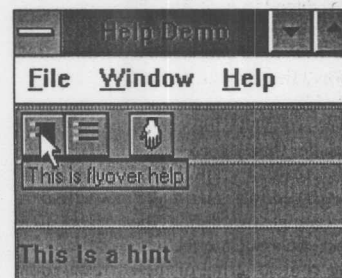


Figure 5: This demo application shows a flyover help box for one of its SpeedBar buttons and a hint line in the status bar at the bottom.

that appears when you rest the mouse cursor on a particular button or other toolbar item for more than a second or two.

Now that you know what elements make up a good help system, you're ready to choose your tools (see the sidebar "Choosing Your Tools") and get to work. With a good design tool and the knowledge you've gained from this article, you should be able to enhance your programs with a thoroughly professional help system. □

NEIL J. RUBENKING IS TECHNICAL EDITOR OF PC MAGAZINE.